Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2019

JOÃO AFONSO AMARAL QUINTELA

Virtual Machine Instantiation Performance Assessment using Virtualization Infrastructure Management

Avaliação de Instanciação de Máquinas Virtuais usando Gestão de Infraestrutura de Virtualização

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2019

JOÃO AFONSO AMARAL QUINTELA

Virtual Machine Instantiation Performance Assessment using Virtualization Infrastructure Management

Avaliação de Instanciação de Máquinas Virtuais usando Gestão de Infraestrutura de Virtualização

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Daniel Nunes Corujo, Investigador Doutorado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Rui Luís Andrade Aguiar, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor António José Ribeiro Neves
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Bruno Miguel de Oliveira Sousa
Professor Auxiliar Convidado da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Doutor Daniel Nunes Corujo
Investigador Doutorado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos / acknowledgements

Agradeço ao Doutor Daniel Corujo toda a orientação dada ao longo da tese. Agradeço aos meus colegas do IT por toda a ajuda e conhecimento partilhado, em particular ao Vítor Cunha e ao Flávio Meneses. Agradecer ao João Verdasca pelas dicas dadas ao longo da tese. Agradecer aos meus pais, amigos e família o apoio dado ao longo dos anos. Dedico este trabalho à minha falecida avó. Esta dissertação foi desenvolvida com o apoio do Instituto de Telecomunicações (UID/EEA/50008/2019) .

Palavras Chave Openstack, 5G, NFV, VIM, VM, SDN, Cloud Computing

Resumo

Duas das tecnologias mais faladas na indústria de redes nos dias de hoje são as Redes Definidas por Software (SDN) e a Virtualização das Funções de Rede (NFV), cujo funcionamento é suportado por tecnologias de gestão de virtualização de infraestruturas. Nesta dissertação, uma das tecnologias usadas foi o Openstack, responsável pela criação e gestão de nuvens públicas e privadas. Neste contexto, esta dissertação apresenta o resultado de uma extensa análise do impacto das características e configurações de diferentes tipos de máquinas virtuais, nos tempos de instanciação associados. Os resultados mostram que, para as várias imagens testadas, os tempos de instanciação diferem tendo em conta o tamanho e a complexidade dessas mesmas imagens.

Keywords Openstack, 5G, NFV, VIM, VM, SDN, Cloud Computing

Abstract Two of the most talked technologies nowadays in the networking industry are Software Defined Networks (SDN) and Network Function Virtualization (NFV), whose functioning is supported by virtualization infrastructure management technologies. In this dissertation, one of the technologies that was studied was Openstack, responsible for the creation and management of public and private clouds. In this context, this dissertation presents an extended analysis of the impact of the characteristics and configurations for different types of virtual machines, in the associated instantiation times. The results show that, for the different tested images, the instantiation times differ with the size and complexity of those images.

Contents

List of Figures

List of Tables

Acronyms

3G Third Generation of Mobile Networks

4G Fourth Generation of Mobile Networks

OSM Open Source Mano

5G Fifth Generation of Mobile Networks

SDN Software Defined Networking

NFV Network Function Virtualization

NFV MANO NFV Management and Orchestration

ETSI European Telecommunications Standards Institute

VIM Virtual Infrastructure Manager

NIST National Institute of Standards and Technology

SaaS Software as a Service

PaaS Platform as a Service

IaaS Infrastructure as a Service

API Application Programming Interface

ONF Open Networking Foundation

ODL OpenDayLight

ONOS Open Network Operating System

OVS Open vSwitch

CAPEX Capital Expenses

OPEX Operating Expenses

ICT Information and Communications Technology

VNF Virtual Network Function

NFVI Network Functions Virtualization Infrastructure

NF Network Function

VM Virtual Machine

VNFM Virtual Network Function Manager

NASA National Aeronautics and Space Administration

LDAP Lightweight Directory Access Protocol

RAM Random Access Memory

CPU Central Processing Unit

KVM Kernel-based Virtual Machine

LXC Linux Containers

REST Representational State Transfer

IP Internet Protocol

API Application Programming Interface

VDI Virtual Desktop Infrastructure

VMDK Virtual Machine Disk

OVF Open Virtualization Format

VHD Virtual Hard Disk

DHCP Dynamic Host Configuration Protocol

MANO Management and Orchestration

NFVO Network Function Virtualization Orchestrator

GUI Graphical User Interface

GB Gigabyte

vCPU virtual CPU

MB Megabyte

Chapter 1

Introduction

This chapter provides the motivation for the development of this dissertation, as well as its objectives. It also describes the document organization.

1.1 Motivation

5G is the fifth generation of mobile technologies, delivering a higher broadband, low latencies and improved networking, when compared to previous mobile generations such as Third Generation of Mobile Networks (3G) or Fourth Generation of Mobile Networks (4G). These features will drastically change not only upcoming technology, but also society, both economically and also the way we live our daily lives. The impact of 5G is so big that it is estimated that in 2035, the full economic benefit of 5G could produce 12.3 trillion dollars worth of goods and support up to 22 million jobs [1].

For the development of 5G, new technologies are starting to emerge. Two of the most important technologies are Software Defined Networking (SDN) and Network Functions Virtualization (NFV). Through a programmable and virtualized network, the combination of SDN and NFV can be vital to future networks, since both of them provide huge scalability and flexibility. With the complete virtualization of networks, the number of users could be massive, transforming the way we see communications throughout the world.

NFV Management and Orchestration (NFV MANO) is a framework developed by ETSI that intends to manage and orchestrate NFV technologies. This framework is divided in 3 blocks that will be detailed later in this thesis. One of those blocks is the Virtualized Infrastructure Manager, also known as Virtual Infrastructure Manager (VIM). VIMs are important for the development of NFV since they are responsible for managing the physical resources capable of deploying network services.

This thesis addresses Openstack, an open-source hypervisor, assessing instantiation times for different instances with different settings. These settings will be modified and then com-

pared to discuss several aspects when instantiating virtual machines.

1.2 Objectives

The main goal for this dissertation is to assess virtual machines instantiation performance using VIM. With this goal in mind, the presented dissertation has the following objectives:

- Study the state of the art for 5G technologies such as NFV MANO.
- Describe Openstack and the impact as a VIM.
- Implement a script that allows to measure instantiation times for different instances.
- Compare instantiation times for different virtual machines.

1.3 Master thesis layout

The remainder of the master thesis is organized as follows:

- Chapter 2 - State of the Art - It discusses the related work and positions the contributions of this work;
- Chapter 3 - Implementation - It provides the implementation and integration of the proposed solutions.
- Chapter 4 - Evaluation - It evaluates the implemented solutions and analyzes the results.
- Chapter 5 - Conclusions - It presents this dissertation's conclusions and future work.

Chapter 2

State of the art

2.1 Introduction

This chapter focuses on providing an overview of the concepts needed to better understand this dissertation. Section 2.2 presents 5G, the fifth generation of mobile technologies and some of the most important technologies to this new generation. Section 2.3 presents the concept of Cloud Computing, as well as some of its key characteristics. Section 2.4 presents SDN, providing its architecture and some of the existing technologies that support SDN. Section 2.5 approaches NFV and the detailed ETSI NFV architecture, also discussing NFV MANO. Section 2.6 discusses Openstack and the most important projects in this software. Finally, Section 2.7 presents the chapter considerations.

2.2 5G

Each generation of mobile networks brings a significant milestone that develops mobile communications and benefits greatly their users, introducing improvements in the network architecture. The fifth generation of mobile networks, unlike previous generations where the main focus was on ensuring connectivity, aims to take mobile networking to the next level, by also providing connected experiences from the cloud to clients [2].

Upcoming 5G network architectures will be mainly defined by software platforms, where network functionalities are entirely managed through software instead of hardware. Figure 2.1 shows some of the key characteristics for 5G.

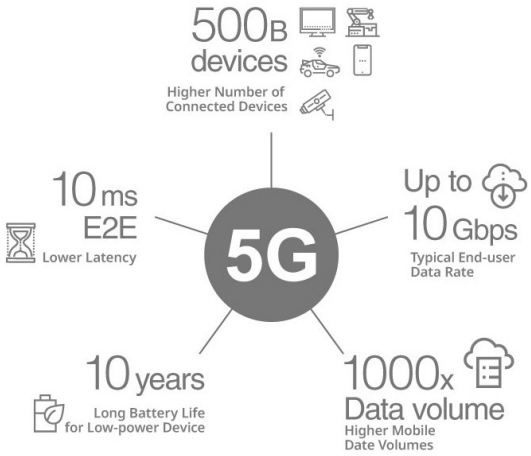

Figure 2.1: 5G key characteristics
[3]

As mentioned previously, 5G technologies have to be able to support a number of services and industries with an enormous diversity of needs and different use cases. The requirements needed to satisfy this amount of services can't be addressed by today's networks. Flexibility and programmability are required to create future virtual networks, and technologies like Software Defined Networks (SDN) and Network Function Virtualization (NFV) can suit the needs of multiple virtual networks for any use case [4].

2.3 Cloud Computing

The name Cloud Computing started to be used when the cloud symbol was popularized to describe Internet. One of the definitions for Cloud Computing was introduced by the National Institute of Standards and Technology (National Institute of Standards and Technology (NIST)): "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models." [5]. In other words, Cloud Computing delivers on-demand computing resources over the Internet, where companies only pay for what they use, providing efficiency in Information Technology (IT) service delivery.

The essential characteristics of Cloud Computing are defined by NIST as follows:

- On-demand self-service - Users have the possibility of automatically reserving resources to their needs, without any sort of human interaction.

- Broad network access - The resources from the cloud are available in the network and their access is possible through mechanisms that support a variety of devices (e.g., mobile clients, laptops, workstations, etc.).

- Resource pooling - Resources must be ready to be accessed by multiple users using a multi-tenant model. The virtual and physical resources must be assigned matching the consumer demand. The user has the perception that the resources available to him are infinite, and does not know the exact location of those same resources.

- Rapid elasticity - Resources must be able to be scaled according to the users needs and must be assigned and/or reassigned dynamically at any time.

- Measured service - Resource usage must be automatically controlled and optimized by cloud systems, providing transparency to both the consumer and the provider of the service.

The Cloud Computing paradigm is also divided in service models. These models can be seen in Figure 2.2. This division is also defined by NIST as follows:

- Software as a Service (Software as a Service (SaaS)) - Allows the user to have the capability of using the provider's applications running on a cloud infrastructure. This cloud infrastructure is all of the hardware and software needed to enable the essential characteristics discussed earlier to deploy Cloud Computing. Users can only adjust some settings related to their specific application.

- Platform as a Service (Platform as a Service (PaaS)) - Users are allowed to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. These users only have the ability of controlling their application, and some other settings where that application is being executed.

- Infrastructure as a Service (Infrastructure as a Service (IaaS)) - IaaS is the capability provided to the consumer to provision key computing resources where the consumer is able to deploy and run software, which can include operating systems and applications. The consumer can control their software, but can only access some settings such as network configurations.

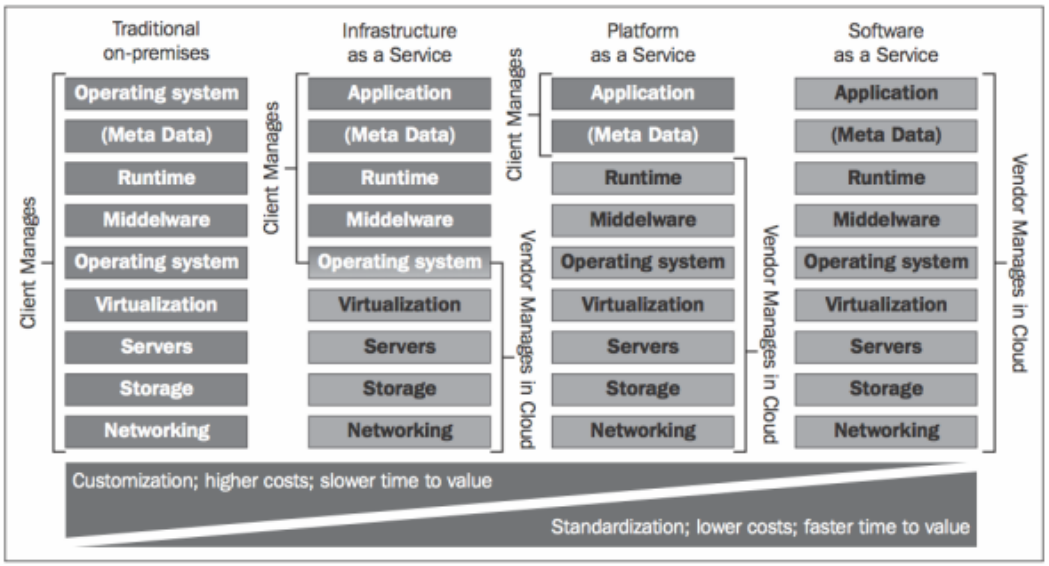

Figure 2.2: Cloud Computing Models
[6]

The Cloud Computing concept was divided by NIST in four different models of deployment, and they are defined as follows:

- Private cloud - A private cloud is built to be used by a single organization/company with multiple users. Only users like employees or managers of those organizations/companies are allowed to access this cloud.
- Community cloud - A community cloud is used by organizations/companies with shared concerns (e.g., mission, security requirements, policy, and compliance considerations).
- Public cloud - A public cloud can be used by the general public, and is usually owned, managed, and operated by a business, academic, or government organization, or some combination of them.
- Hybrid cloud - Lastly, the hybrid cloud is the combination of two or more clouds that were talked previously. This cloud infrastructure can then be private or public, according to the specified settings.

2.4 SDN

As discussed previously, 5G network architectures will be mainly defined by software platforms. One of the key building blocks for these networks is Software-Defined Networking (SDN). SDN is a dynamic and adaptable architecture that decouples the network control

plane from the data/forwarding plane, turning the network into an entity that is completely programmable. The decoupling of these two planes allows networks to be more manageable and agile, where the behavior of the network equipments is entirely controlled through a logically centralized SDN controller [7]. All of these features allow networks to better adapt to more dynamic environments, something important for future 5G architectures [8].

2.4.1 SDN Architecture

The SDN architecture is built to centralize the network intelligence, abstracting the network infrastructure from the applications, due to the fact that the control plane is decoupled from the data/forwarding plane. This decoupling provides a higher scalability, increasing network control, and allowing enterprises and carriers to build extremely flexible networks [9].

This architecture is visible in figure 2.3, and is divided by 3 different layers: the application layer, the control layer (control plane), and the infrastructure layer (data plane). There are also two types of interfaces: the northbound interface and the southbound interface. The northbound interface is located between the application layer (north) and the control layer, and the southbound interface is located between the control layer and the infrastructure layer (south).

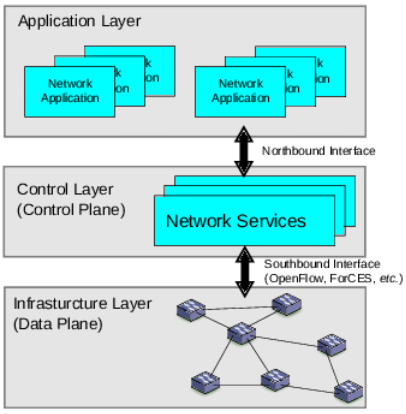

Figure 2.3: SDN architecture and layers
[10]

2.4.1.1 Application layer

The application layer is where services and applications are located. SDN applications define the network behavior, and provide end-to-end solutions for enterprises and data centre networks. These applications communicate through the application-controller plane interface, both network requirements and some management functions [11].

2.4.1.2 Control layer

The control layer is responsible for making the applications requests, deciding how packets are forwarded by network devices. This layer also gives constant updates of the routing table, and does the configuration and management of the system. Both the northbound and southbound interface are connected to the control plane. The northbound interface communicates with the application layer and the protocols used for this interface are Representational State Transfer (REST)ful Application Programming Interface (API)s of SDN controllers. The southbound interface communicates with the infrastructure layer, and is realized with southbound protocols, such as Openflow or Netconf [12].

2.4.1.3 Infrastructure layer

The infrastructure layer (data/forwarding plane) is the layer that handles traffic. The data and control planes are decoupled, meaning that the control plane gives network administrators the opportunity to easily manage how routers and switches of the data plane handle the users traffic. Through the southbound interface, communication between the control plane and data plane is made by protocols. Some of the southbound protocols used are Openflow or NetConf.

2.4.2 SDN Projects

- Openflow - Openflow is the flagship of the Open Networking Foundation (Open Networking Foundation (ONF)), and is widely regarded by many as the first SDN standard. Openflow was originally created in 2008 for researchers, and with the hability of running experimental protocols in various networks. Openflow is used as the interface between the control layer and network devices (infrastructure layer), providing a centralized overview of all the events that occur throughout the network [13].

- OpenDayLight - The OpenDayLight project (OpenDayLight (ODL)) was created by the Linux Foundation. The main goal of this open-source project is to enhance SDN, deploying a SDN controller, that allows users to customize the controller to their specific needs. In another words, ODL allows their users to deploy SDN as they please, reducing operational complexity and enabling new services and capabilities [14].

- Open Network Operating System (ONOS) - The Open Network Operating System (ONOS) is an open-source project hosted by Linux Foundation. ONOS is a SDN controller designed to meet the requests of operators, providing both configuration and real-time control of the network [15].

- Floodlight - The Floodlight SDN controller is a Java-based Openflow controller. This controller is good for developers for two reasons: the controller is written in Java and software is easy to adapt, working in various environments [16].

- Open vSwitch - Open vSwitch (OVS) offers a virtual switch, with Openflow being the protocol used to control traffic. OVS is key in SDN deployments, as it offers a virtual switch that ties together all the Virtual Machine (VM)s within a hypervisor instance on a specific server [17].

2.5 NFV

The main idea of NFV is to virtualize network functions, decoupling network functions from dedicated hardware such as routers or firewalls and enabling those functions on VMs (Virtual Machines). This provides a higher degree of flexibility, given that network functions can now be instantiated at anytime [18].

As network functions are virtualized, NFV has the opportunity to reduce both Capital Expenses (CAPEX) (Operating Expenses) and Operating Expenses (OPEX) (Operating Expenses), facilitating the deployment of new services and also create favorable OPEX and CAPEX prospects. This happens because companies don't have to purchase proprietary hardware, which means that time-to-market is greatly reduced [19].

NFV shows some differences in which network services provisioning is done in comparison to current practices. ETSI, also known as European Telecommunication Standards Institute, is an independent standardization organization, responsible for developing standards for information and communications technologies (Information and Communications Technology (ICT)) in Europe [20]. The differences mentioned above, where listed by ETSI as follows:

- Decoupling software from hardware: As the network elements are no longer a composition of integrated hardware and software entities, the evolution of both is independent of each other. This enables the software to progress separately from the hardware, and vice versa.

- Decoupling software from hardware: As the network elements are no longer a composition of integrated hardware and software entities, the evolution of both is independent of each other. This enables the software to progress separately from the hardware, and vice versa.

- Dynamic scaling: The decoupling of the functionality of the network function into instantiable software components provides greater flexibility to scale the actual Virtual Network Function (VNF) performance in a more dynamic way and with finer granularity,

for instance, according to the actual traffic for which the network operator needs to provision capacity.

Even though NFV presents a number of advantages by running functions with virtualized resources (e.g flexibility, scaling), it's important to point out that it is still possible that functions are running on virtualized resources and physical resources at the same time.

2.5.1 ETSI NFV/Architecture

ETSI defined the NFV architecture with 3 key elements: NFV Management and Orchestration (NFV MANO), Network Functions Virtualization Infrastructure (Network Functions Virtualization Infrastructure (NFVI)) and the VNFs and services [21]. These elements are shown in Figure 2.4.

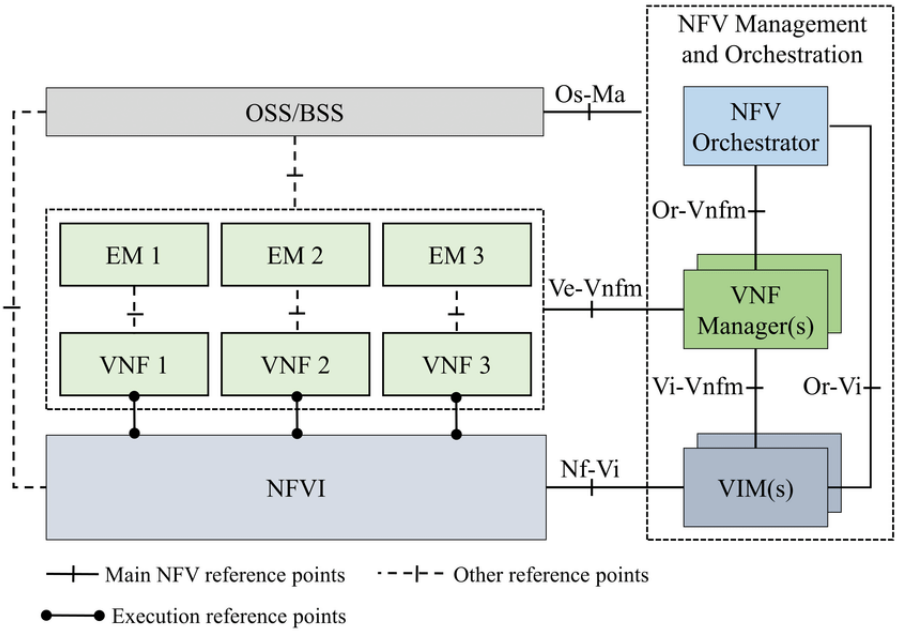

Figure 2.4: ETSI-NFV architectural framework
[22]

2.5.1.1 Network Functions Virtualization Infrastructure

NFVI is the combination of software and hardware resources needed to create an environment capable of running VNFs. Many of the VNFs are in the form of virtual machines, and this is where it can be found physical servers, storage, networking, and the virtualization layer (also known as hypervisors). Physical resources consist of computing hardware, storage and network that provide processing, storage and connectivity to VNFs. As for the

virtual resources, they are an abstraction of computing, storage and network resources. The hypervisors mentioned previously are used to virtualize network functions, decoupling these functions from software/hardware they run on [23].

2.5.1.2 Virtual Network Functions and Services

To better understand what a Virtual Network Function (VNF) is, we must look first at what is a network function (Network Function (NF)). A NF is a functional block that composes a network infrastructure, with well-defined external interfaces and functional behaviour. Viewing a network function in a practical way, a NF is often a network node. A VNF is nothing more than a NF that is deployed with virtual resources. VNFs can be deployed in several VMs, and inside each VM is hosted a single component of the VNF. It is also possible that the entire VNF is deployed in one single VM [24].

2.5.1.3 Network Function Virtualization Management and Orchestration

The NFV MANO is responsible for providing the functionalities to provision and configure VNFs. NFV MANO has 3 main blocks, each of them with a different interaction with VNFs: Network Function Virtualization Orchestrator (Network Function Virtualization Orchestrator (NFVO)), Virtual Network Function Manager (Virtual Network Function Manager (VNFM)) and Virtual Infrastructure Manager (VIM) [25]. The blocks are described as follows:

- NFVO - The NFVO is responsible for orchestrating and managing the lifecycle of the NFVI and the virtual and physical resources.

- VNFM - VNF Manager performs the management of the life-cycle of VNFs (e.g. instantion, scaling, update, termination, etc). VNFM can be deployed to one or multiple VNF instance(s).

- VIM - VIM is a specific block responsible for the control and management of the NFVI resources such as compute, storage or network. VIMs can be instantiated in a network multiple times, managing failure information, capacity planning and optimization.

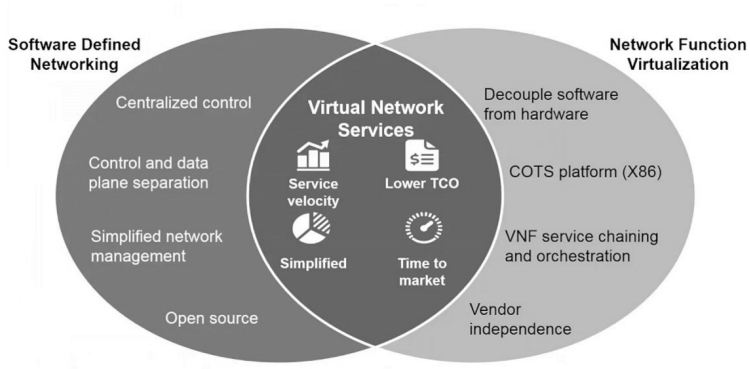

Figure 2.5: SDN and NFV combined
[26]

2.5.2 Open Source Mano

Open Source Mano (OSM), better known as Open Source Mano, delivers an ETSI-hosted open source management and orchestration (Management and Orchestration (MANO)) community project, where the primary goal is to provide a MANO stack that pretends to meet the requirements specified by commercial NFV networks [27].

OSM intends to follow ETSI NFV reference architecture in terms of features and building blocks. As talked about previously, ETSI NFV architectural framework is constituted by two very important components: the NFV orchestrator and the VNF Manager. Through projects like OSM, open-source software eases the implementation of ETSI architectures aligned with NFV. As ETSI OSM is well complemented by ETSI NFV, this allows for a higher level of innovation and maximizes time to market. OSM is currently on release 6. This recent release has brought newer capabilities, a better management of complex services and as a wider range of technologies supported by this open-source software.

2.6 Openstack

Openstack is an open-source software used to build and manage computing platforms for public, private and hybrid clouds. It is owned by Openstack Foundation, an entity that provides the support needed to keep Openstack running and performing. This open source software was originally designed by National Aeronautics and Space Administration (NASA) and Rackspace [28].

Openstacks's official projects are written in Python and they can be run in Linux systems. In reality, Openstack is not a software in itself, but more a set of components and services, where each has a well defined functionality.

This open source software is defined by a series of projects that control large pools of

computing, storage, and also network resources in a data center while managing through a dashboard. Openstack has several core components, each of them important for the modular architecture that defines this software:

- Horizon - Horizon is a simple modular web interface, providing a user interface for cloud infrastructure management, allowing administrators and users to access a graphic interface easy to deploy.

- Keystone - This is a centralized service that provides a way of account management, authentication and permission of access to services. It is integrated with other authentication services such as Lightweight Directory Access Protocol (LDAP) (Lightweight Directory Access Protocol), authentication multi-factor, token authentication and even Kerberos.

- Nova - This component is responsible for managing the processing resources of the platform such as Random Access Memory (RAM) or Central Processing Unit (CPU). Nova is key to manage the life-cycle of the users VMs, controlling the creation, execution and termination of each VM through drivers that communicate with the virtualization layer where we can find hypervisor technologies such as Kernel-based Virtual Machine (Kernel-based Virtual Machine (KVM)), VMware solutions, Xen, Hyper-V and Linux Containers (LXC).

- Glance - Glance does the image management in Openstack, allowing the registration of new images and providing those same images to Nova to create VMs. These images can be viewed as templates to create new VMs. Glance has a user REST API that does the search and transfer of images when VMs are created. Glance supports a lot of image formats, such as Raw, VirtualBox (V (VDI)), VMWare (Virtual Machine Disk (VMDK), Open Virtualization Format (OVF)), Hyper-V (Virtual Hard Disk (VHD)), and Qemu/KVM (qcow2) virtual machine images.

- Neutron - To manage networks and Internet Protocol (IP) addresses in Openstack, Neutron is the component responsible for that. Neutron allows users to create IP networks with diverse topologies, connecting each of the VMs and delivers management IP services such as Dynamic Host Configuration Protocol (Dynamic Host Configuration Protocol (DHCP)). Devices and servers can then be connected to one or more networks.

- Swift - Swift is a storage system highly scalable that deploys data and non-structured objects through an API RESTful. Swift has the ability of saving and replicating information through various servers, providing high data redundancy.

- Cinder - OpenStack Cinder delivers a block-storage service that is responsible for the management of virtual hard drives within the infrastructure. Cinder does the creation,

association and dissociation of blocks. The entire management of information can be managed by the user, integrating block storage volumes with Dashboard and Nova.

- Ceilometer - This component as emerged with Openstack Havana and it deploys services to manage several infrastructures. Ceilometer is responsible for monitoring notifications from services that already exist, so that any sort of developer can collect and configure data as they want.

- Heat - Heat implements an orchestration engine to multiple composite cloud applications based on templates in the form of text files. To orchestrate the applications, Heat uses both an OpenStack-native REST API and a CloudFormation-compatible Query API.

2.6.1 Devstack

Devstack is a series of scripts that offers a complete Openstack environment, with smaller system requirements and easier to deploy. The main purpose is to develop Openstack in a single machine in a faster way, allowing users to develop solutions on top of it. Devstack makes significant changes to the system in the installation process, and should only be run on servers or virtual machines that are exclusively dedicated to it [29].

2.7 Chapter considerations

This chapter provided the state of the art for the remainder of the thesis. It was provided an overall view on 5G, and introduced some concepts such as Cloud Computing, SDN and NFV. For SDN and NFV, both their architectures were described and detailed for a better understanding of the impact that they will have in upcoming 5G network architectures. Openstack was also presented, as well as the most important projects of this software.

Chapter 3

Implementation

3.1 Introduction

This chapter describes with more detail the implementation to measure the VM instantiation delay in Openstack. Section 3.2 describes the Openstack services most used and how they interact with each other. Section 3.3 describes how an instance is launched in Openstack. Section 3.4 provides the system specification and settings.Section 3.5 characterizes the network necessary to measure the instantiation times. Section 3.6 describes the script and the method to calculate instantiation times. Section 3.7 presents the chapter summary and considerations.

3.2 Openstack Services

Although Openstack services have been previously discussed in the chapter "State of the Art", it is important to detail how these services work and interact between each other, in particular, when an instance is launched in Openstack. The services that are going to be explained in further detail are: Nova, Neutron, Keystone, Glance and Horizon.

3.2.1 Openstack Nova

Openstack Nova, also known as Openstack Compute, is a component developed for users to create, manage, and destroy virtual servers. In another words, Nova provides access on-demand to computing resources, while provisioning and managing large networks of virtual machines. This Openstack project supports the creation of virtual machines, baremetal servers and has limited support for system containers. Nova is engaged with virtualization technologies such as VMware, KVM, Hyper-V Hypervisor, Xen, Virtuozzo, and Linux container technologies (more specifically LXD and LXC). Most Openstack Nova installations

only use one hypervisor. These hypersivors are supported by Openstack Nova, and can be defined as follows:

- VMWare vSphere - vSphere Hypervisor is a bare-metal supervisor used to virtualize servers. This virtualization platform (formerly known as VMWare Infrastructure) includes two main components: VMWare ESXi and the VMWare vCenter server. ESXi is a type 1 (bare-metal) hypervisor that abstracts multiple resources into virtual machines. As for the VMWare vCenter server, this server allows to manage the entire infrastructure of the vSphere. VMWare vSphere runs VMware-based Linux and Windows images through a connection to the vCenter server.

- Virtuozzo - This platform provides a hypervisor, capable of hosting virtual machines. Virtuozzo provides an OS virtualization layer that is responsible for providing support for running containers. These containers are easy to use, managing physical resources much more dinamically. Figure 3.1 shows some aspects of the Virtuozzo architecture.

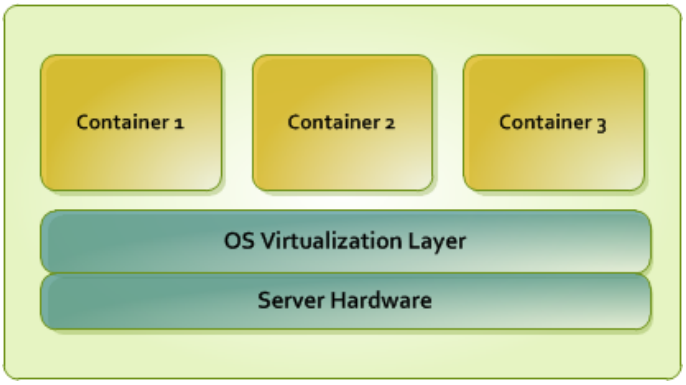

Figure 3.1: Virtuozzo OS Virtualization
[30]

- Xen - The Xen Project is a type 1 (bare-metal) hypervisor, available as an open-source software. This hypervisor uses libvirt as a management interface into Openstack Nova, running virtual machines such as Linux, Windows, FreeBSD and NetBSD. Xen is unique because the hypervisor doesn't have any device drivers, keeping domains/guests isolated. In Xen, "Dom0", or Domain 0, is the initial and privileged domain started by the Xen hypervisor on boot. This domain is responsible for managing the "DomU" unprivileged domains. Figure 3.1 presents the XEN architecture.

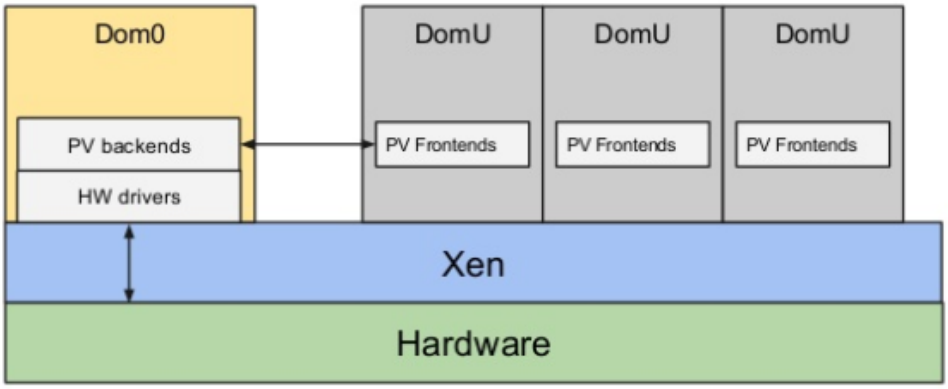

Figure 3.2: XEN Architecture
[31]

- Hyper-V - Hyper-V is a type-1 native hypervisor designed by Microsoft, used to run Windows, Linux, and FreeBSD virtual machines. The hypervisor, a core component of Hyper-V, is a layer of software between the hardware and the operating system. The hypervisor allows multiple systems to run without interfering with the processes in other virtual machines. In Figure 3.3, the Hyper-V architecture is represented.

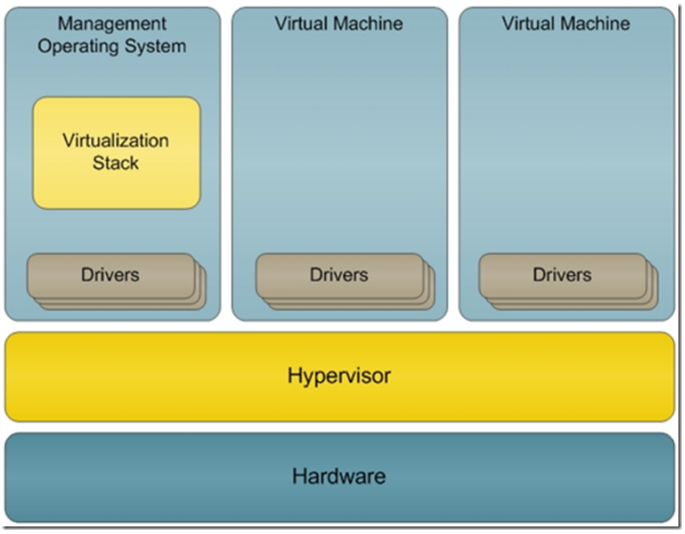

Figure 3.3: Hyper-V Architecture
[32]

- KVM - KVM is a full virtualization platform for Linux on x86 hardware containing virtualization extensions. This technology converts Linux into a type-1 hypervisor. KVM is a part of the Linux kernel, with Red Hat Enterprise Linux (RHEL) including

KVM since version 5.4. With KVM, it is possible to run multiple virtual machines with Linux or Windows images. Each virtual machine has private virtualized hardware like network cards, disks and graphics adapters.

- LXC - LXC represents Linux Containers, that are used to run Linux-based virtual machines. This virtualization platform is much different than the other platforms previously discussed, because LXC works at the operating system level. This means that other containers may affect the resources of other containers that are hosted on the same virtual machine.

Nova's architecture is defined by a distributed application with several components. These components are mostly Python daemons of two types: WSGI (Web Server Gateway Interface) applications that manage API calls and daemons responsible for carrying out orchestration tasks. To function in the most basic way, Nova requires additional Openstack services such as Keystone, Neutron and Glance, which will be explained in further detail later on.

Openstack Nova architecture can be seen in Figure 3.4:

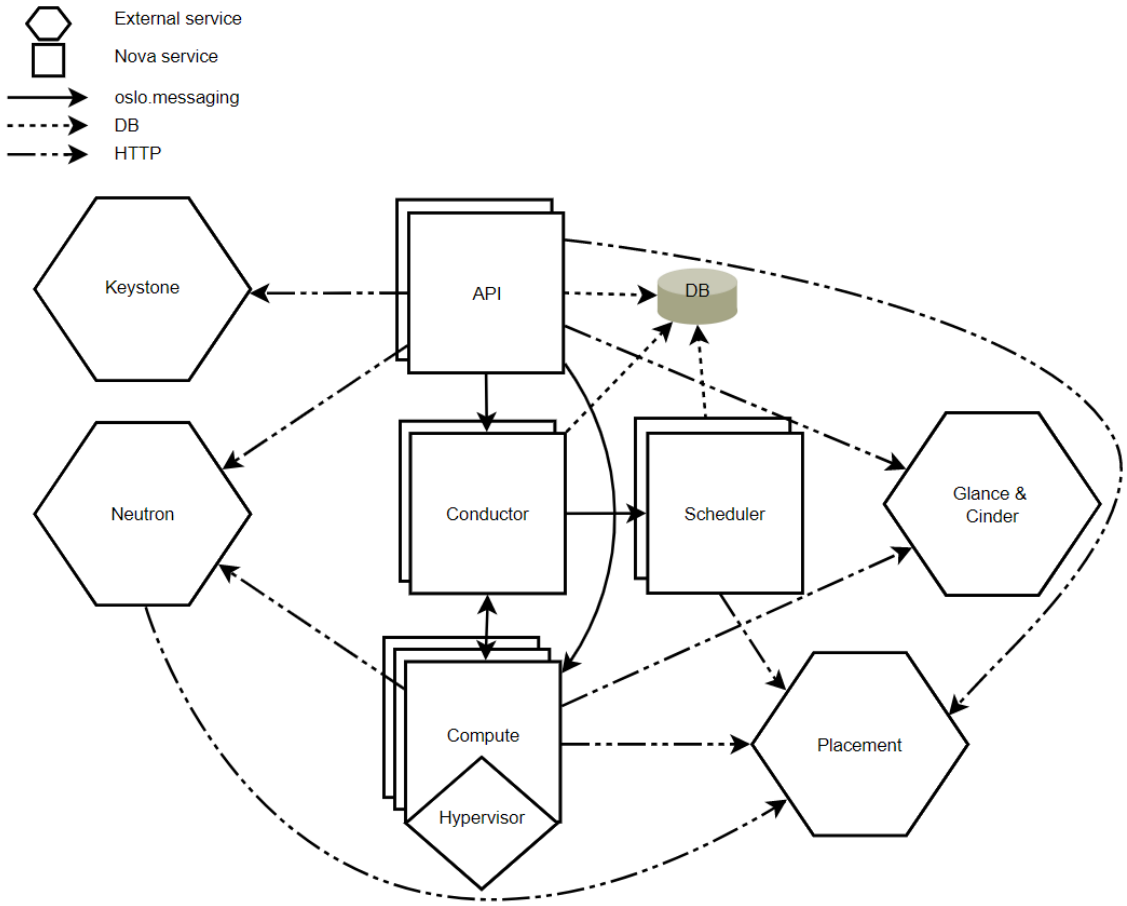

Figure 3.4: Openstack Nova Architecture
[33]

Looking at Nova's architecture in more detail, it is possible to explain the interaction between each block:

- DB (Database) - The database is responsible for storing most of the configurations and run-time state for the cloud infrastructure. These configurations includes all sorts of information, such as the networks available, the projects, instances used and instance types ready to be deployed.

- API - The API in control of nova is the nova-api. This component receives and accepts HTTP requests, communicating with other components through oslo.messaging queue;

- Compute - This component creates and terminates virtual machine instances. The management of virtual machines is made most usually through a toolkit that manages virtualization platforms, called libvirt and created by Red Hat;

- Scheduler - The scheduler takes a virtual machine instance request from the queue and decides which host should run it;

- Conductor - Conductor handles requests to be coordinated when it comes to build or resize, providing database access support for compute nodes, which reduces security risks.

3.2.2 Openstack Neutron

Openstack Neutron is behind the creation and management of virtual networking infrastructures in the Openstack cloud. This project is a SDN project that delivers NaaS (Networking as a Service), making networking services available for Openstack Nova. Neutron provides network objects such as networks, subnets or routers, allowing other services within Openstack to use Neutron through the API. The networking topologies to be created may include services such as a firewall, a load balancer and VPN's (Virtual Private Networks).

Networking in Openstack has to have at least one internal network and one external network. The external network is not defined virtually, being accessible outside the Openstack installation, as well as IP addresses. Software defined networks connect directly to virtual machines, which means that only VMs on any given internal network can have access to virtual machines that are connected to that network.

Another aspect to take into consideration are the security groups supported by Neutron. Administrators of the cloud can define firewall rules in groups. For a virtual machine, it is possible to have more than one security group, applying the defined firewall rules to that same VM. Figure 3.5 shows the Openstack Neutron networking setup:

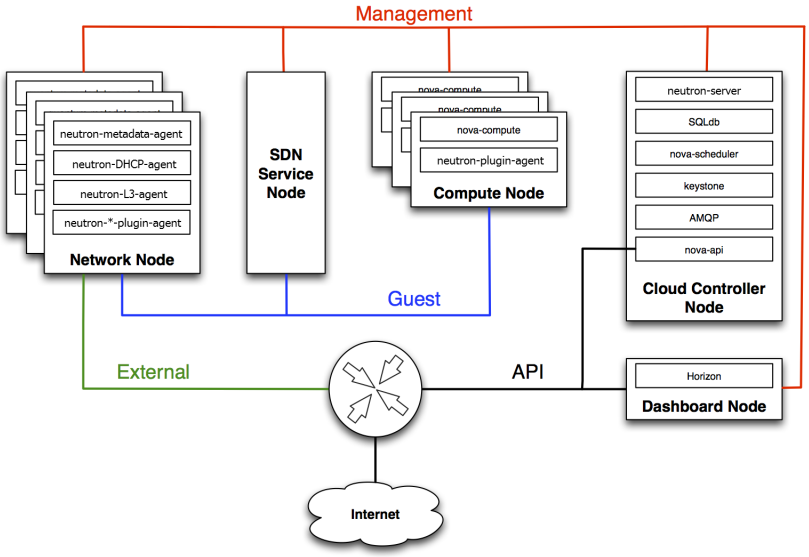

Figure 3.5: Openstack Neutron networking setup
[34]

Looking at the previous figure, any standard networking setup in Neutron has 4 different physical data center networks:

- Management network - Manages the internal communication between Openstack components.
- Guest network - Used for VM data communication within the cloud deployment.
- External network - Gives virtual machines access to the internet in some deployment scenarios.
- API network - Exposes Openstack API's to tenants.

3.2.3 Openstack Keystone

Keystone is Openstack's identity service. It is used for both authentication (authN) and high-level authorization (authZ). In Keystone, the authentication service uses a combination of domains, projects, users, and roles. For that, it is worthy to understand the relation between users, projects, groups and domains:

- Users - Person, system or service represented digitally.
- Groups - A number of users.
- Projects - Groups used to isolate resources and/or users.

- Domains - A number of users, groups, and projects

Taking in consideration the previous definitions, it is possible to establish a connection on how they interact with each other, as seen in Figure 3.6:

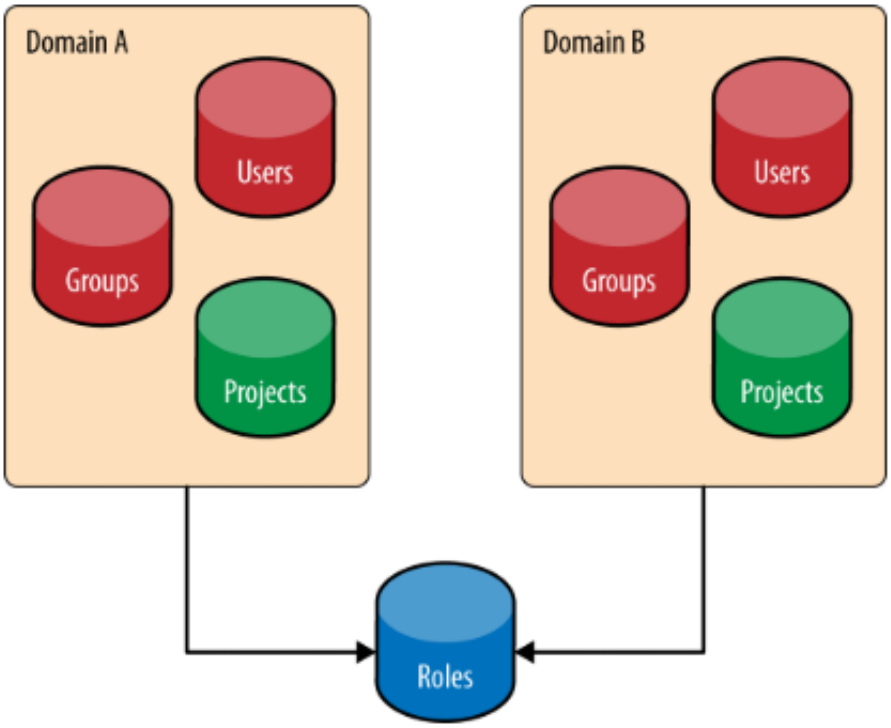

Figure 3.6: Interaction between Users, Groups, Projects and Domains [35]

As previously mentioned, roles are used to authenticate services in Keystone. Roles are a set of assigned user privileges to perform any specific operation. However, to authenticate and authorize interactions between Openstack APIs, tokens are used. Tokens are bits of text used to access resources with a variety of scopes and multiple sources of identity. These tokens are then used to provide information about a user's role assigments.

3.2.4 Openstack Glance

The image service of Openstack is Glance, and is responsible for enabling users to manage virtual machine images. Glance does the storage, registration and retrieve of virtual machine images. These virtual machine images are files that contain a virtual disk with a bootable operating system installed.

Openstack Nova makes use of Glance while an instance is being provisioned, as all compute instances launch from Glance images. In Figure 3.7, is the logical architecture of Openstack

Glance.

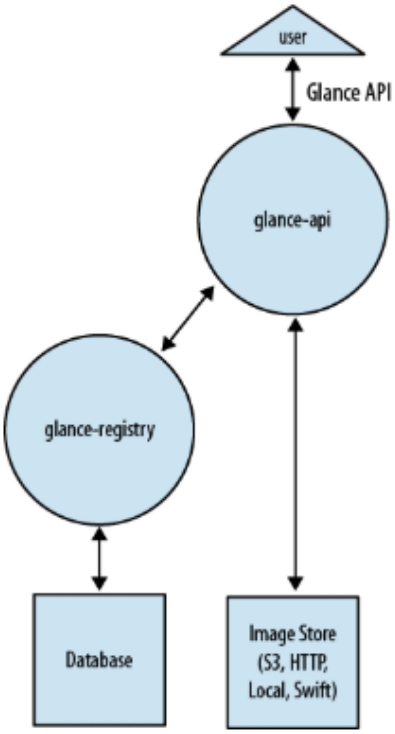

Figure 3.7: Openstack Glance Logical Architecture
[36]

In the architecture we can see 2 blocks that are important to a better understanding of Glance: glance-api and glance-registry. glance-api accepts API calls, where as glance-registry stores and retrieves the images metadata.

Glance supports a variety of disk formats. The image's disk format can be set in Glance to one of the following:

- Raw - Unstructured disk image format.
- VHD - Disk format used by virtual machine monitors from VMware, Xen, Microsoft, VirtualBox, etc.
- VHDX - Enhanced version of the VHD format which supports larger disk sizes.
- VMDK - Disk format that describes containers to be used in virtual machine monitors.
- VDI - Disk format supported by VirtualBox virtual machine monitor and the QEMU emulator.
- ISO - Disk image of an optical disk.

- Ploop - Disk format supported and used by Virtuozzo to run OS Containers.
- qcow2 - Disk format supported by the QEMU emulator.

3.2.5 Openstack Horizon

Horizon is the graphical interface of Openstack's Dashboard, providing a web-based user interface for users to access and manage Openstack services. Figure 3.8 shows the Openstack Dashboard.

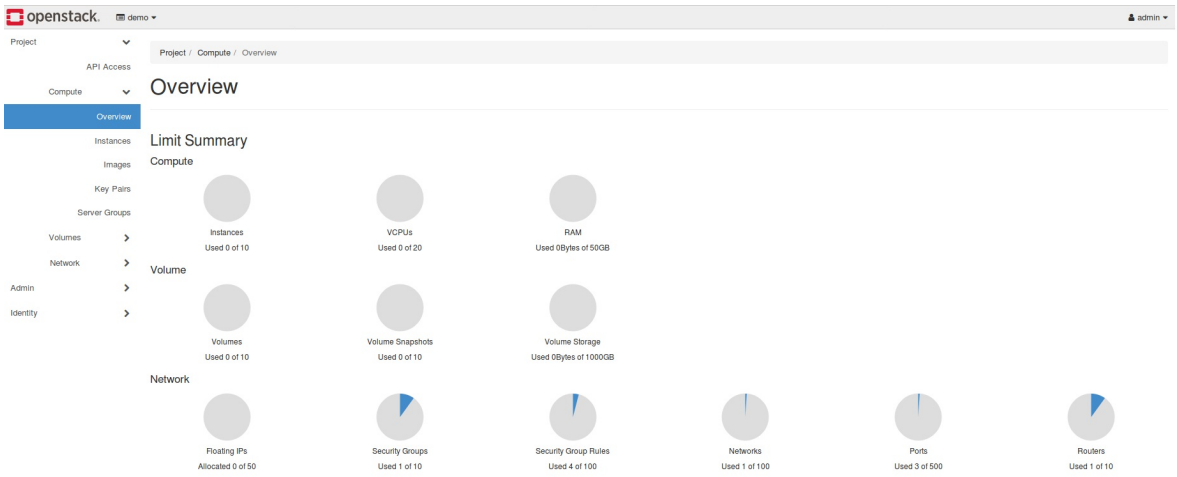

Figure 3.8: Openstack Dashboard

Openstack Horizon offers 3 central dashboards: a "User Dashboard", a "System Dashboard" and also a "Settings Dashboard". Administrators of the cloud can make any sort of changes in some of the visual elements of Horizon. Horizon needs some basic requirements in order to be installed: Python (version 2.7) and Django (version 1.7 or 1.8), Openstack Nova, Openstack Neutron, Openstack Keystone and Openstack Glance.

3.3 Instances

In Openstack, instances are virtual machines that run inside the cloud. The reason why in Openstack virtual machines are called instances is due to the fact that these virtual machines are in fact instances of an image that is created when requested and configured when its launched. An important aspect of Openstack is that it supports both ephemeral and persistent models, unlike other virtualization technologies. In persistent models, the persistent volume is not launched from the image service, but is launched instead from a compute node or a block storage volume. As for the ephemeral models, the instance is launched from an image in the image service (Glance), then that image is copied to the area where the instance will be

run, starting the instantiation. The advantage of the ephemeral model is that the scalability is much quicker, granting users a high agility.

To launch an instance in Openstack, several settings have to be specified such as the instance name, flavors, images, networks and security groups. To create an instance, it is important to check if everything is ready for the instance to get started, such as the network, key pair and an image or a volume as the boot source.

3.3.1 Flavors

Flavors are used to define the compute, memory, and storage capacity of nova computing instances. Flavors are responsible for defining the size of a virtual server that can be launched.

The key parameters for flavors are: flavor ID, name, vCPUs, memory (in MB) and the disk (in GB). Table 3.1 lists some of the default flavors provided by Openstack:

Flavor	vCPUs	Total Disk (GB)	RAM (MB)
m1.nano	1	1	64
m1.micro	1	1	128
cirros256	1	1	256
m1.tiny	1	1	512
ds512M	1	5	512
ds1G	1	10	1024
m1.small	1	20	2048
ds2G	2	10	2048
m1.medium	2	40	2048
ds4G	4	20	4096
m1.large	4	80	8192
m1.xlarge	8	160	16384

Table 3.1: Openstack default flavors

3.3.2 Images

To use a virtual machine image in Openstack, the easiest way to do so is to download one that someone else has already created. The test image for Openstack is CirrOS. CirrOS offers a minimal and simple linux distribution designed for use as a test image on clouds such as OpenStack Compute.

There are several virtual machine images that can be used in Openstack. The virtual machine images tested (besides CirrOS) are the following:

- CentOS - CentOS is an open-source Linux distribution launched in 2014 and is a replica of the Red Hat Enterprise Linux (RHEL). Similarities with RHEL make CentOS great for businesses and developers. The lack of community support and more difficult installation makes CentOS tricky to pick up for Linux beginners. CentOS latest version is CentOS version 8. The versions to be used in this dissertation will be CentOS 6 and CentOS 7.

- Debian - Debian is a Linux distribution constituted by free and open-source software. The first stable version of Debian was launched in 1996. Debian has 3 branches of releases: a stable release branch, a testing release branch and an unstable release branch. These branches allow users to choose the branch that suits their needs better.

- Ubuntu - Ubuntu is a free open-source Linux distribution. The first release of the Ubuntu distribution was in 2004, and it was based on the Debian system. Ubuntu provides a distro that is easy to deploy, given the fact that this Linux distribution doesn't require user configuration during the installation process, unlike Debian.

- Arch-Linux - Arch-Linux, also known as Arch, offers a Linux distribution for computers that are based on x86-64 architectures. This distribution is lightweight and flexible, and based in binary packages.

The images installed in the Openstack Dashboard are shown in Figure 3.9.

Name ▲	Type	Status	Visibility	Protected	Disk Format	Size
Arch-Linux-x86_64	Image	Active	Public	No	QCOW2	1.30 GB
CentOS-6-x86_64	Image	Active	Public	No	QCOW2	769.38 MB
CentOS-7-x86_64	Image	Active	Public	No	QCOW2	898.75 MB
cirros-0.4.0-x86_64-disk	Image	Active	Public	No	QCOW2	12.13 MB
Debian-10-amd64	Image	Active	Public	No	QCOW2	540.19 MB
Debian-9-amd64	Image	Active	Public	No	QCOW2	564.90 MB
Ubuntu-16.04-x86_6	Image	Active	Public	No	QCOW2	328.44 MB

Figure 3.9: Images in Openstack Dashboard

3.3.3 Security groups

By using Neutron, it is possible to configure security groups for the instance that will be deployed. Security groups in Openstack limit any sorts of traffic, and can be created when

an instance is launched. When creating these security groups, one or more can be assigned, but if no security group is created, the instance will run the default security group.

In this case, the security groups that are going to be created are TCP for SSH and ICMP for pings. TCP (Transmission Control Protocol) is used to allow access to instances through SSH (Secure Shell). The standard TCP port for SSH is 22. As for the ICMP security group, the main purpose is to ping the instances to be deployed.

3.4 Evaluation Approach

The main objective of this dissertation is to perform evaluation tests on different virtual machines. With that being said, the conclusions that will be withdrawn and the way tests are going to be performed, is through the instantiation times of different virtual machines.

The images to be tested and compared, when it comes to instantiation times are the folowing:

- CirrOS - x86_64
- Debian 9 - amd64
- Debian 10 - amd64
- CentOS 6 - x86_64
- CentOS 7 - x86_64
- Ubuntu 16.04 - x86_64
- Arch Linux - x86_64

With the instantiation times that will be taken from these images, assumptions can be made about the complexity and size of the respective virtual machine.

3.5 System specification and settings

The computer used to perform the tests for this dissertation, has the following specifications: Intel Core i7-8550U Processor 1.8GHz Quad-core, 16GB RAM 1866MHz LPDDR4 and a 512GB SATA3 M.2 SSD.

In that computer, a virtual machine was created. The virtual machine that was chosen was Oracle VM VirtualBox with the following specifications: 10 GB of RAM, 2 vCPU's and 70 GB of disk (virtual size).

While creating the virtual machine, ubuntu-16.04.6-server was installed. The choice to install ubuntu-server instead of ubuntu-desktop was due to the fact that the system chosen is

much lighter, something important when it comes to the measurement of instantiation times, specially for Openstack and the amount of services that this software runs.

Another aspect to take into consideration are the ports that need to be specified in the VirtualBox. These ports will enable the communication with Horizon and enable SSH. The ports configured for this work are represented in Figure 3.10.

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Horizon	TCP		8080		80
SSH	TCP		2222		22

Figure 3.10: Port Forwarding Rules

After the configuration of the virtual machine, the next step was to install Devstack. As mentioned previously, Devstack is a series of scripts that offers a complete Openstack environment, with smaller system requirements and easier to deploy. The main purpose is to develop Openstack in a single machine in a faster way, allowing users to develop solutions on top of it.

Finished the installation, Devstack installed keystone, glance, nova, placement, cinder, neutron, and horizon. Through the access to horizon, it is possible to open the web interface to Openstack. In this web interface, VMs, networks, volumes and images can be managed.

In Figure 3.11 is represented the conceptual architecture for Openstack, where the services talked previously are represented.

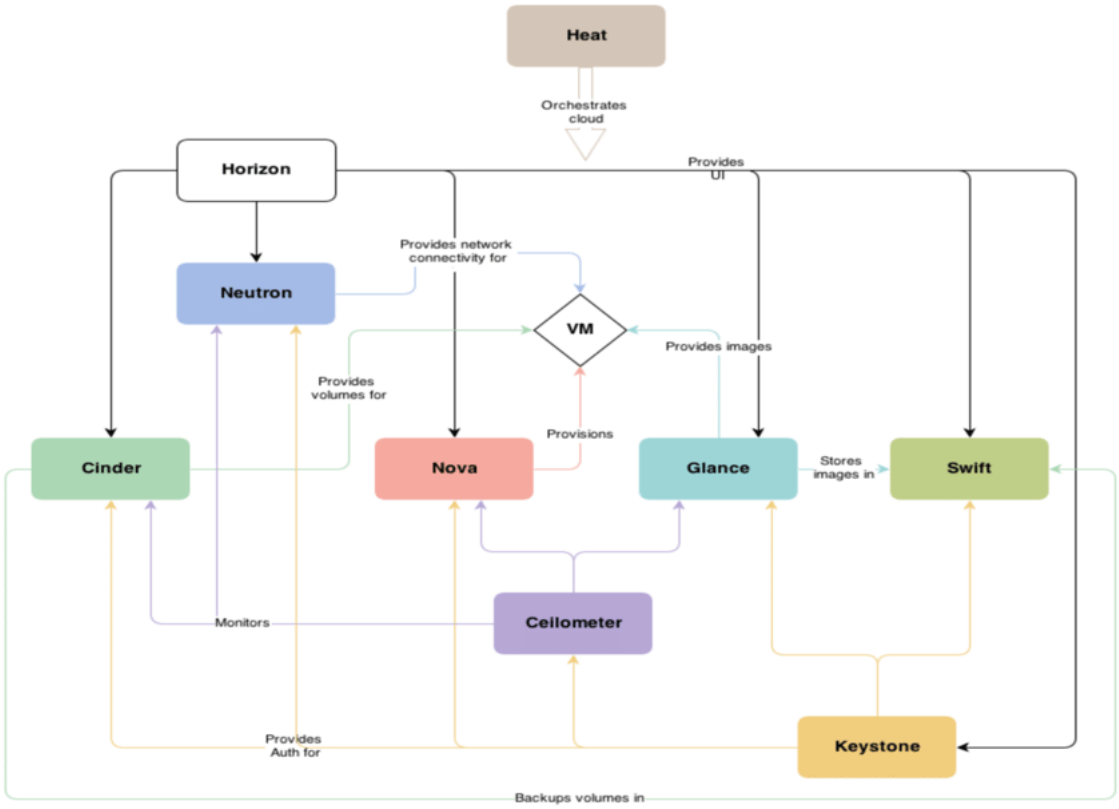

Figure 3.11: Openstack Conceptual Architecture
[37]

3.6 Network

First, we need to create the network where the instantiation will occur. For that, we must set the environment variables using the Openstack RC File. The Openstack RC File is an environmental file that sets the environment variables for any Openstack command-line clients. This file can be downloaded from the Openstack Dashboard for users with administrative privileges.

After sourcing the RC File, these environment variables ensure that Openstack client commands are communicating with the Openstack services that run in the cloud.

To create the network, we start by creating a private network through the command line. There are several private network blocks available to use with private networks. These private networks are in CIDR format. CIDR stands for Classless Inter-Domain Routing, a method to allocate both IP addresses and IP routing. The most common example for this is 10.0.0.0/24, the one that is used in this implementation.

After the private network is created, there is a new network with a subnet to be specified. A subnet is then created and accepted by the network previously created. The next step is

to create a router with a default configuration. This router will then connect to the subnet, connection that is made trough the command line once again. The only thing left to do is set the router gateway to the public network. The private network is now successfully created. The network topology to be used in this dissertation is expressed in Figure 3.12.

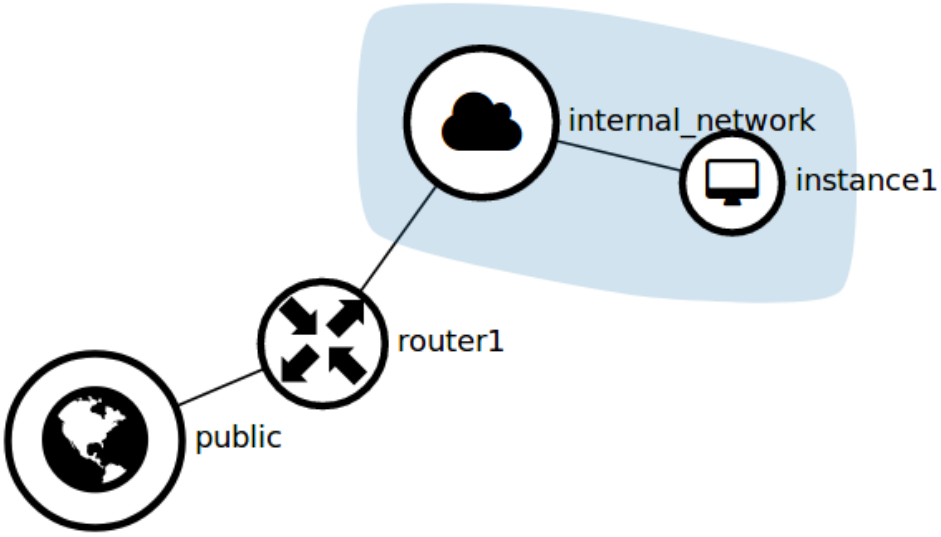

Figure 3.12: Network Topology

3.7 Instantiation time

For that purpose, it was created a script called cherry.py, a script based on CherryPy. CherryPy is an object-oriented web application framework that uses Python. CherryPy allows developers to build web applications in much the same way they would build any other object-oriented Python program. It is designed for a rapid-application development of web applications.

To calculate the initial instantiation time, the script cherry.py must be run. This will give a time that will be stored as a time stamp. This time stamp provides the initial time that the script communicates with Openstack Horizon, thus, the initial instantiation time. Afterwards, another script must be run called openstack_commands. This script automatically creates the network where the instance will occur. In this openstack_commands script, these are the events that occur:

- Creation of the security groups that will enable pings and SSH (protocols TCP and ICMP).

- Upload the ssh key.

- Creation of the private network.

- Create of the subnet within the private network.

- Create a new router.

- Connect the router to the subnet and to the gateway named public.

- Create and then boot the instance.

- Generate the floating IP (public IP address) to the instance.

- Add the floating IP to the instance.

When the virtual machine is instantiated, a wget must be done, that will communicate with the cherry.py's web service, that will store the stop time in a file named times.csv, calculating the final instantiation time for the virtual machine. The diagram of all the steps to get the instantiation time is depicted in Figure 3.13.

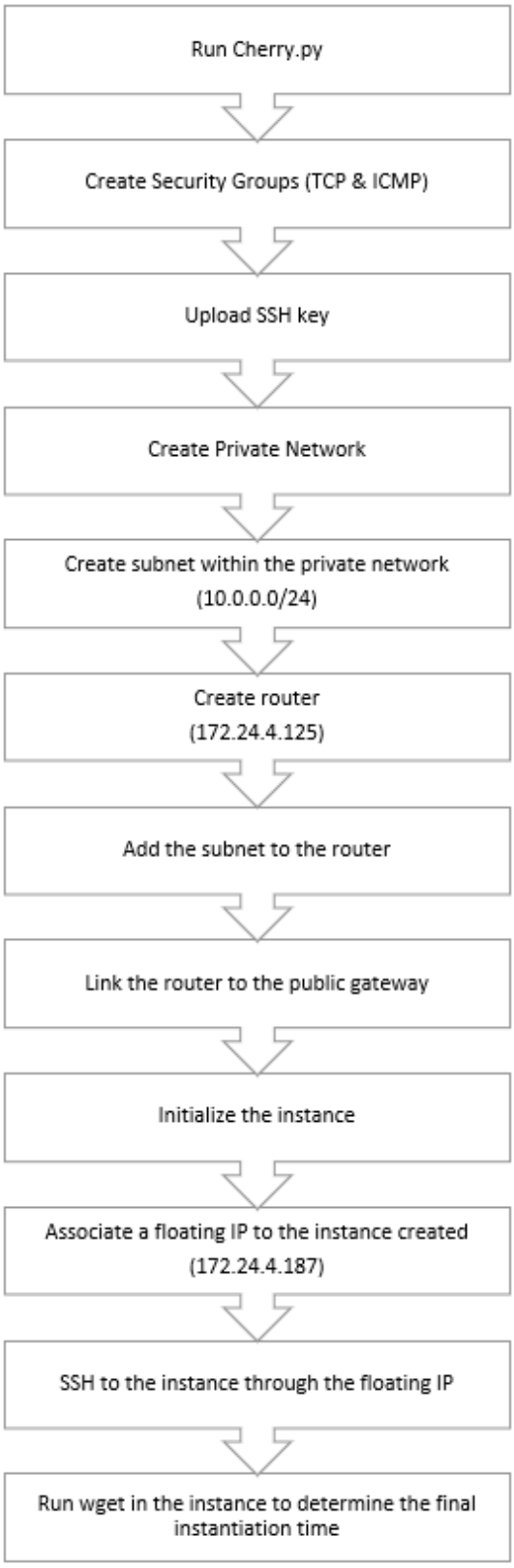

Figure 3.13: Steps to get the instantiation time

In order to see what should be done in order to prove that everything is working properly, we first must check the connectivity of the VM to the network that was created. The ping was first tested to the router of the VM. The IP address for the router was 172.24.4.125. This ping was successfull, and is represented in Figure 3.14:

```
PING 172.24.4.125 (172.24.4.125) 56(84) bytes of data.
64 bytes from 172.24.4.125: icmp_seq=1 ttl=64 time=0.115 ms
64 bytes from 172.24.4.125: icmp_seq=2 ttl=64 time=0.098 ms
64 bytes from 172.24.4.125: icmp_seq=3 ttl=64 time=0.111 ms
64 bytes from 172.24.4.125: icmp_seq=4 ttl=64 time=0.125 ms
64 bytes from 172.24.4.125: icmp_seq=5 ttl=64 time=0.141 ms
```

Figure 3.14: Ping to the router

After successfully connecting to the router of the created network, the next step was to ping the floating IP associated to the instance. The instance used had flavor m1.tiny, with 1 VCPUs, 512MB of RAM and 1GB of GB. For this instance, the image given was cirros-0.4.0-x86. In this example, the floating IP associated to the instance tested was 172.24.4.187 and the instance tested was CirrOS. After associating the floating, the ping is successful, has shown in Figure 3.15:

```
PING 172.24.4.187 (172.24.4.187) 56(84) bytes of data.
64 bytes from 172.24.4.187: icmp_seq=1 ttl=63 time=5.44 ms
64 bytes from 172.24.4.187: icmp_seq=2 ttl=63 time=1.10 ms
64 bytes from 172.24.4.187: icmp_seq=3 ttl=63 time=0.853 ms
64 bytes from 172.24.4.187: icmp_seq=4 ttl=63 time=1.19 ms
```

Figure 3.15: Ping to the floating IP associated to the instance

After pinging the floating IP we can now access through SSH to the instance. The scripts to calculate the instantiation time can be seen in the Appendix A for the Openstack commands and in the Appendix B for the cherry.py script.

3.8 Chapter considerations

In this chapter it was discussed how the implementation to achieve results for the virtual machine instantiation performance was made. We started by analyzing Openstack services in more detail. Then, looking at what are instances in Openstack and some of the details on how they are launched. Afterwards, the system specification and settings to start the testing were presented. The network where instances will be launched was also presented. And finally, the procedure and explaining of the scripts to calculate instantiation times was also presented.

Chapter 4

Evaluation

4.1 Introduction

This chapter describes the results obtained from the evaluations performed. Section 4.2 presents the results for CirrOS. Section 4.3 approaches the results for Debian 9. Section 4.4 presents the instantiation times for Debian 10. Section 4.5 provides the results for CentOS 6. Section 4.6 presents the results for CentOS 7. Section 4.7 evaluates the instantiation time for Ubuntu 16.04. And finally section 4.8 presents the results for Arch Linux.

4.2 CirrOS

For CirrOS, the image used was the most recent 64-bit qcow2 image, which is cirros-0.4.0, with a size of 12.13 MB. The flavors used to determine the instantiation time were cirros256 (256 MB of RAM, 1 GB of disk and 1 vCPU), m1.small (2048 MB of RAM, 20 GB of disk and 1 vCPU) and ds1G (1024 MB of RAM, 10 GB of disk and 1 vCPU).

4.2.1 cirros256

Average instantiation time (s)	103.07
Standard deviation (%)	1.24

For the CirrOS image and flavor cirros256, the lowest instantiation time was 101 seconds, and the highest instantiation time was 105 seconds. The average instantiation time was 103.07 seconds, with a standard deviation of 1.24 %, meaning the results obtained are within a good range of values.

4.2.2 m1.small

Average instantiation time (s)	102.47
Standard deviation (%)	1.16

The second flavor used to calculate the instantiation time with CirrOS was m1.small. The highest instantiation time was 104 seconds, and the lowest 101 seconds, meaning the average instantiation time was 102.47 seconds. The standard deviation was 1.16 %, a value that, once again, means that the range of values doesn't change considerably. Another aspect to be observed is the average instantiation time, that is similiar to the value for the flavor cirros256.

4.2.3 ds1G

Average instantiation time (s)	102.33
Standard deviation (%)	1.02

The third and final flavor tested for the CirrOS image, was ds1G. For this flavor, the highest instantiation time was 104 seconds, and the lowest 101 seconds, meaning the average instantiation time was 102.33, with a standard deviation of 1.02 %. As for the other two flavors, the instantiation times were extremely close.

4.2.4 CirrOS summary

For the CirrOS image, the average instantiation times for the flavors cirros256, m1.small and ds1G were respectively 103.07, 102.47 and 102.33 seconds. This means that, regardless of the flavor, the instantiation time won't vary much. CirrOS is a minimal Linux distribution, hence the small size of the Cirros image. It is expected that instantiation times for the remainder of the images will be higher, as the distributions to be instantiated are more complex and have a bigger image size.

4.3 Debian 9

For Debian 9, the image used was Debian-9-amd64, with size 564.90 MB. The flavors used to determine the instantiation time were m1.small (2048 MB of RAM, 20 GB of disk and 1 vCPU) and ds1G (1024 MB of RAM, 10 GB of disk and 1 vCPU).

4.3.1 m1.small

Average instantiation time (s)	193.4
Standard deviation (%)	2.00

The first flavor to be tested in Debian 9 was m1.small. The highest instantiation time was 199 seconds and the lowest instantiation time was 188 seconds. As for the average instantiation time, the value was 193.4 seconds. The standard deviation was 2.00 %. It is noticeable that the instantiation time is, as expected, much higher for a Debian 9 image than for a CirrOS image.

4.3.2 ds1G

Average instantiation time (s)	191.07
Standard deviation (%)	1.68

The second and final flavor tested for Debian 9 was ds1G. The highest value for the instantiation time was 196 seconds and the lowest value was 186 seconds. The average instantiation time was 191.07 seconds and the standard deviation was 1.68 %.

4.3.3 Debian 9 summary

For the Debian 9 image, the instantiation time was significantly higher when compared to the CirrOS image. This can be explained with the fact that the image size and the complexity of the image of Debian 9 being bigger than the CirrOS image. As for the different flavors testes, once again, the values did not differed a lot between the flavors.

4.4 Debian 10

For Debian 10, the image used was Debian-10-amd64, with size 540.19 MB. The flavors used to determine the instantiation time were m1.small (2048 MB of RAM, 20 GB of disk and 1 vCPU) and ds1G (1024 MB of RAM, 10 GB of disk and 1 vCPU).

4.4.1 m1.small

Average instantiation time (s)	191.8
Standard deviation (%)	1.34

For the first measure of the instantiation time with the image of Debian 10 and flavor m1.small, the highest instantiation time was 196 seconds and the lowest value was 187 seconds, with an average instantiation time of 191.8 seconds. The standard deviation in percentage was 1.34 %. These values are similar to the ones obtained for Debian 9.

4.4.2 ds1G

Average instantiation time (s)	191.2
Standard deviation (%)	1.69

The values for instantiation times for flavor ds1G were similar to the ones registered for flavor m1.small. In this case, the highest instantiation value was 196 seconds and and the lowest was 187 seconds, with an average instantiation time of 191.2 seconds. The standard deviation value was 1.69 %.

4.4.3 Debian 10 summary

For the flavors m1.small and ds1G, the values were very similar, meaning that instantiation times are not dependent of different flavors. For Debian 10, the average instantiation times were almost the same, and these values were identical to the ones previously taken for Debian 9. Not only is the image of Debian 9 similar to Debian 10 (they share the same Linux distribution), the size and details of both images is similar, resulting in identical instantiation times.

4.5 CentOS 6

For CentOS 6, the image used was CentOS-6-x86_64, with size 769.38 MB. The flavors used to determine the instantiation time were m1.small (2048 MB of RAM, 20 GB of disk and 1 vCPU) and ds1G (1024 MB of RAM, 10 GB of disk and 1 vCPU).

4.5.1 m1.small

Average instantiation time (s)	207.27
Standard deviation (%)	1.80

For CentOS 6, the flavor m1.small was used to calculate the instantiation time. The highest value was 213 seconds and the lowest value was 201 seconds, with an average instantiation time of 207.27 seconds and a standard deviation of 1.80 %.

4.5.2 ds1G

Average instantiation time (s)	207.67
Standard deviation (%)	2.21

The second flavor used in CentOS 7 was ds1G. The highest value for the instantiation was 217 seconds and the lowest value was 200 seconds, with an average instantiation time of 207.67 seconds. The standard deviation was a little bit higher than usual, at about 2.21 %.

4.5.3 CentOS 6 summary

CentOS 6 was so far the image that took the longest to instantiate. The flavors had again very close values, but the instantiation time had the biggest value so far. Not only is the

CentOS 6 image the biggest image tested so far (769.38 MB), but it also provides a complex Linux distribution, where the instance takes longer to start.

4.6 CentOS 7

For CentOS 7, the image used was CentOS-7-x86_64, with size 898.75 MB. The flavors used to determine the instantiation time were m1.small (2048 MB of RAM, 20 GB of disk and 1 vCPU) and ds1G (1024 MB of RAM, 10 GB of disk and 1 vCPU).

4.6.1 m1.small

Average instantiation time (s)	210.07
Standard deviation (%)	2.67

For CentOS 7, the first flavor used was m1.small. The biggest value for the instantiation time was 218 seconds and the lowest value was 203 seconds. The average instantiation time was 210.07 seconds, (tooking longer than CentOS 6 to start the instance) with a standard deviation of 2.67 %.

4.6.2 ds1G

Average instantiation time (s)	210.47
Standard deviation (%)	2.22

For the second flavor tested (ds1G) the average instantiation time was 210.47 seconds, with values ranging from 204 to 218 seconds. The standard deviation was 2.22 %.

4.6.3 CentOS 7 summary

Comparing the performance between the 2 CentOS images, the instantiation time for CentOS 7 is slightly bigger, because not only is the image size bigger for CentOS 7, the degree of complexity is more significant in the more recent release. For these reasons, CentOS 7 took the longest to instantiate, even though the instantiation was only a bit bigger than in CentOS 6.

4.7 Ubuntu 16.04

For Ubuntu 16.04, the image used was Ubuntu-16.04-x86_6, with size 328.44 MB. The flavors used to determine the instantiation time were m1.small (2048 MB of RAM, 20 GB of disk and 1 vCPU) and ds1G (1024 MB of RAM, 10 GB of disk and 1 vCPU).

4.7.1 m1.small

Average instantiation time (s)	201.73
Standard deviation (%)	1.55

For the Ubuntu image and using the m1.small flavor, the average instantiation time was 201.73 seconds, ranging from 198 to 208 seconds. The standard deviation in the measurements was approximately 1.55 %.

4.7.2 ds1G

Average instantiation time (s)	201.27
Standard deviation (%)	1.39

As for the flavor ds1G, the highest instantiation value was 206 seconds and the lowest instantiation value was 197 seconds, with an average instantiation time of 201.27 seconds. The standard deviation was 1.39 %.

4.7.3 Ubuntu 16.04 summary

Looking at the results for Ubuntu 16.04, the instantiation time was actually higher than for the Debian image for example. Although Ubuntu has a smaller size image, the number of libraries and frameworks that need to be deployed to start an Ubuntu instance is much higher when compared to Debian. It is also worth to mention that Ubuntu is based in Debian.

4.8 Arch Linux

For Arch Linux, the image used was Arch-Linux-x86_64, with size 1.3 GB. The flavors used to determine the instantiation time were m1.small (2048 MB of RAM, 20 GB of disk and 1 vCPU) and ds1G (1024 MB of RAM, 10 GB of disk and 1 vCPU).

4.8.1 m1.small

Average instantiation time (s)	230.47
Standard deviation (%)	2.61

The first flavor analyzed for the instantiation of Arch Linux was m1.small. The average instantiation time for the Arch Linux image was approximately 230.47 seconds with a standard deviation of 2.61 %.

4.8.2 ds1G

Average instantiation time (s)	228.53
Standard deviation (%)	3.20

The last instantiation was for the same Arch Linux image but now with a ds1G flavor. The highest instance time was 241 seconds while the lowest tume was 218 seconds. The average instantiation time was 228.53 seconds with a standard deviation of 3.20 %.

4.8.3 Arch Linux summary

Arch Linux image was the one that had the longest average instantiation time. The main reason for this is the enormous size of the image, that has 1.3 GB. The amount of time it takes to instantiate Arch Linux is normal for the size at hands.

4.9 Comparison between images

To better look and compare images between each other, Table 4.1 contains the average instantiation times for the two flavors that were used (m1.small and ds1G):

Images	m1.small	ds1G
CirrOS	102.47	102.33
Debian 9	193.4	191.97
Debian 10	191.8	191.2
CentOS 6	207.27	207.67
CentOS 7	210.07	210.47
Ubuntu 16.04	201.73	201.27
Arch Linux	230.47	228.53

Table 4.1: Average instantiation times for m1.small and ds1G flavors (in seconds)

In Figures 4.1 and 4.2 are represented the average instantiation times for the m1.small and ds1G flavors respectively.

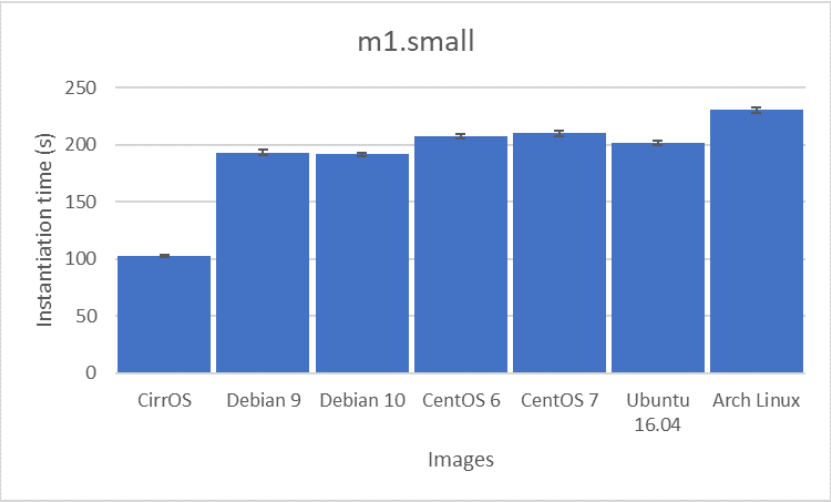

Figure 4.1: Comparison between images for m1.small

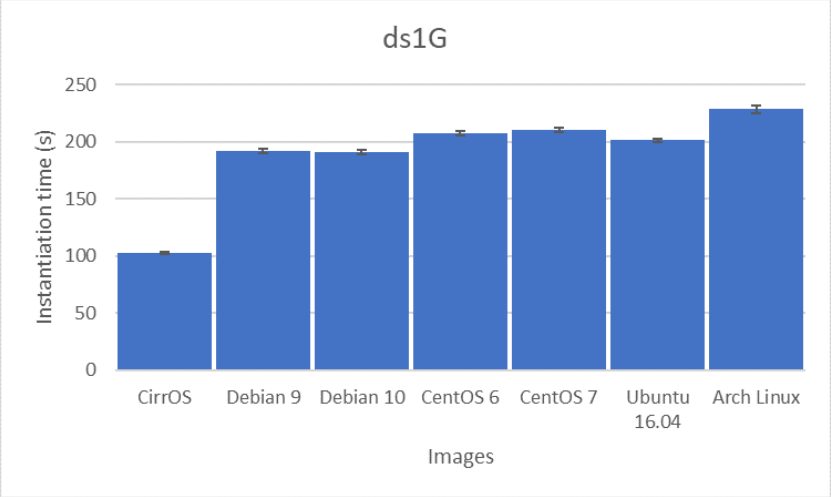

Figure 4.2: Comparison between images for ds1G

The smallest instantiation time was for CirrOS and the biggest instantiation time was for Arch Linux. The main aspect to take into consideration when looking into aspects that can have influence in instantiation times, the image size is the most important characteristic. The CirrOS image is extremely small and provides a minimalist Linux distribution, where Arch Linux has a much larger size, making the instantiation time bigger in consequence.

A curious result was the average instantiation time of Ubuntu 16.04 when compared to Debian. Ubuntu is based on Debian, and the image size for Ubuntu is actually smaller than the image size of Debian, However, Ubuntu includes everything in their default repository, making it as easy and as accessable to the user.

4.10 Chapter considerations

In this chapter, the results for the instantiated machines were analyzed, taking into consideration different flavors and other aspects. Next chapter presents the thesis' conclusions and future work

Chapter 5

Conclusions

5.1 Conclusion

This thesis presented an assessment on the performance of virtual machine instantiation using virtualization infrastructure management.

Openstack is a free and open-source software platform for cloud computing, where is possible to build and manage private and public clouds. In Openstack, instances are virtual machines that run inside the cloud and in order to launch an instance, parameters such as the security groups, flavors of the instance and images need to be gathered before the lauching. This dissertation presented an extended analysis of the impact of the characteristics and configurations for different types of virtual machines and their associated instantiation times.

Through Openstack, images (virtual machines) were launched inside the cloud. The images tested were: CirrOS, Debian 9, Debian 10, CentOS 6, CentOS 7, Ubuntu 16.04 and Arch Linux. These images all had in common that, with the change of flavors, there was no significant change in the instantiation times. However, the size of the images launched had quite an impact in the instantiation of images.

CirrOS was the the image that took less to be launched, mainly because of its size and minimalistic Linux Distribution. As for CentOS (6 and 7) and also Arch Linux, the opposite happened, as they where the images that took the longest to be instanced. The complexity of these distributions and the size of these images where the key factors as to why they where the longest instantiation times.

Another aspect to look to is the comparison between Ubuntu and Debian (9 and 10). Even though the image size for Debian is bigger, Ubuntu's Linux distribution is much more complex than Debian's distribution, having many more software aggregated when comparing to Debian.

5.2 Future work

As a future work for this dissertation, more refined solutions could be made in order to better assess the performance for virtual machine instantiation.

Another aspect to evaluate would be the performance for other VIMs in the market and also other images.

Bibliography

[1] Qualcomm, *Everything You Need to Know About 5G*, 2019. https://www.qualcomm.com/invention/5g/what-is-5g.Lastaccessed11June 2019.

[2] NGMN Alliance, *5G White Paper*, 2015. Last accessed 11 June 2019.

[3] Gigabyte. https://www.gigabyte.com/Article/what-you-must-know-before-moving-into-5g . Last accessed 21 September 2019.

[4] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J.J. Ramos-Munoz, J. Lorca, J. Folgueira, *Network Slicing for 5G with SDN/NFV: Concepts, Architectures and Challenges*, 2017. Last accessed 11 June 2019.

[5] Peter Mell and Timothy Grance, *The NIST Definition of Cloud Computing*, 2011. Last accessed 11 June 2019.

[6] E. Schouten. https://edwinschouten.nl/2014/02/07/cloud-computing-defined-characteristics-service-levels/ . Last accessed 21 September 2019.

[7] M. Rouse, *SDN controller (software-defined networking controller)*, 2018. https://searchnetworking.techtarget.com/definition/SDN-controller-software-defined-networking-controller. Last accessed 14 June 2019.

[8] N. Cranford, *The role of NFV and SDN in 5G*, 2017. https://www.rcrwireless.com/20171204/fundamentals/the-role-of-nfv-and-sdn-in-5g-tag27-tag99. Last accessed 14 June 2019.

[9] M. Raza, *What is Software Defined Networking? SDN Explained*, 2018. https://www.bmc.com/blogs/software-defined-networking/. Last accessed 14 June 2019.

[10] https://www.researchgate.net/figure/A-three-layer-software-defined-networking-SDN-architecture_fig1_284696928 . Last accessed 21 September 2019.

[11] H. Arora, *Software Defined Networking (SDN) - Architecture and role of OpenFlow*, 2018. https://www.howtoforge.com/tutorial/software-defined-networking-sdn-architecture-and-role-of-openflow/. Last accessed 14 June 2019.

[12] S. Staff, *What are SDN Southbound APIs?*, 2014. https://www.sdxcentral.com/networking/sdn/definitions/southbound-interface-api/. Last accessed 14 June 2019.

[13] Noviflow, *The basics of SDN and the OpenFlow Network Architecture*, 2018. https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/. Last accessed 14 June 2019.

[14] OpenDayLight. https://www.opendaylight.org/. Last accessed 14 June 2019.

[15] B. O'Connor. https://www.opennetworking.org/onos. Last accessed 14 June 2019.

[16] ProjectFloodlight. http://www.projectfloodlight.org/floodlight/. Last accessed 14 June 2019.

[17] OpenvSwitch. https://www.openvswitch.org/. Last accessed 15 June 2019.

[18] CISCO, *NFV - Network Functions Virtualization*, 2019. https://www.cisco.com/c/en/us/solutions/service-provider/network-functions-virtualization-nfv/index.html. Last accessed 15 June 2019.

[19] *5 Ways Network Function Virtualization (NFV) Lowers CapEx and OpEx.* https://www.itbusinessedge.com/slideshows/5-ways-network-function-virtualization-nfv-lowers-capex-and-opex.html. Last accessed 16 June 2019.

[20] ETSI, *NFV IN ETSI*, 2019. https://www.etsi.org/technologies/nfv. Last accessed 16 June 2019.

[21] M. Abu-Lebdeh, *ETSI NFV architectural framework*, 2017. https://www.researchgate.net/figure/ETSI-NFV-architectural-framework_fig1_320956787. Last accessed 17 June 2019.

[22] https://www.researchgate.net/figure/ETSI-NFV-architectural-framework_fig1_320956787. Last accessed 21 September 2019.

[23] S. Staff, *What Is NFV Infrastructure (NFVI)*, 2016. https://www.sdxcentral.com/networking/nfv/definitions/nfv-infrastructure-nfvi-definition/. Last accessed 17 June 2019.

[24] *What is a Virtual Network Function or VNF?*, 2019. https://www.sdxcentral.com/networking/nfv/definitions/virtual-network-function/. Last accessed 17 June 2019.

[25] F. Khan, *A Beginner's Guide to NFV Management Orchestration (MANO)*, 2015. https://www.telcocloudbridge.com/blog/a-beginners-guide-to-nfv-management-orchestration-mano/. Last accessed 17 June 2019.

[26] http://www.netsegypt.com/insights/sdn_nfv/index.html. Last accessed 21 September 2019.

[27] OSM, 2019. https://osm.etsi.org/. Last accessed 3 July 2019.

[28] Rackspace, *What is OpenStack?*, 2019. https://www.rackspace.com/library/what-is-openstack. Last accessed 18 June 2019.

[29] *Devstack*, 2019. https://docs.openstack.org/devstack/latest/. Last accessed 25 June 2019.

[30] https://docs.virtuozzo.com/virtuozzo_7_users_guide/learning-virtuozzo-basics/os-virtualization-layer.html. Last accessed 21 September 2019.

[31] https://www.slideshare.net/linaroorg/lcu14-308-xen-project-for-arm-servers. Last accessed 21 September 2019.

[32] https://blogs.technet.microsoft.com/tonyso/2009/04/02/hyper-v-how-to-understand-hyper-v-architecture/. Last accessed 21 September 2019.

[33] https://docs.openstack.org/nova/stein/user/architecture.html. Last accessed 21 September 2019.

[34] https://ilearnstack.wordpress.com/tag/quantum/. Last accessed 21 September 2019.

[35] https://www.oreilly.com/library/view/identity-authentication-and/9781491941249/ch01.html. Last accessed 21 September 2019.

[36] https://www.oreilly.com/library/view/deploying-openstack/9781449311223/ch03.html. Last accessed 21 September 2019.

[37] https://steemit.com/utopian-io/@khatab505/understanding-and-architecture-of-openstack. Last accessed 21 September 2019.

Appendix A - Openstack commands

```
# Security group that will enable inbound ping & SSH
openstack security group create --description "ssh & icmp enabled"
admin-ssh

openstack security group rule create --protocol tcp --dst-port
22:22 --remote-ip 0.0.0.0/0 admin-ssh

openstack security group rule create --protocol icmp admin-ssh

# SSH key
ssh-keygen -b 2048 -t rsa -f admin-key -P ""
openstack keypair create --public-key admin-key.pub admin-key

# Private network
openstack network create admin-net
openstack network list

# Subnet within the private network
openstack subnet create --network admin-net
--subnet-range 10.0.0.0/24 admin-subnet1

openstack subnet list

# Router
openstack router create admin-router

# Connect the subnet to the router
openstack router add subnet admin-router admin-subnet1
```

```
# Connect the router to the gateway public
openstack router set --external-gateway public admin-router
openstack router show admin-router

# Check flavors
openstack flavor list

# Create/boot of the instance
openstack server create instance1 \
--flavor flavor_to_add \
--image image_to_add \
--key-name admin-key \
--security-group admin-ssh \
--nic net-id=admin-net

# Floating IP address for the instance
openstack floating ip create public
```

Appendix B - Cherry script

```
import cherrypy

import time

import subprocess

config = {
        'global' : {
                'server.socket_host' : '0.0.0.0',
                'server.socket_port' : 7000
        }
}

tstart = time.time()

class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        out = open('/opt/stack/times.csv', 'a')
        tstop = time.time()
        it_time = tstop-tstart

        out.write(str(int(tstart)) + ', ' +  str(int(tstop)) + ', '
        + str(int(it_time)) + '\n')

        out.flush()
        return "Final countdown"

cherrypy.quickstart(HelloWorld(), '/',  config)
```